

WEST[Help](#)[Logout](#)[Interrupt](#)[Main Menu](#) | [Search Form](#) | [Posting Counts](#) | [Show S Numbers](#) | [Edit S Numbers](#) | [Preferences](#) | [Cases](#)**Search Results -**

Terms	Documents
L7 and table	28

Database:

US Patents Full-Text Database
US Pre-Grant Publication Full-Text Database
JPO Abstracts Database
EPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search:

L8 and

[Refine Search](#)

[Recall Text](#)  [Clear](#)

Search History**DATE:** Saturday, December 13, 2003 [Printable Copy](#) [Create Case](#)**Set Name** **Query**
side by side**Hit Count** **Set Name**
result set*DB=USPT; PLUR=NO; OP=OR*

<u>L8</u>	L7 and table	28	<u>L8</u>
<u>L7</u>	L6 and call	28	<u>L7</u>
<u>L6</u>	L4 and table	35	<u>L6</u>
<u>L5</u>	L4 and lookup ADJ table	0	<u>L5</u>
<u>L4</u>	L3 and table	35	<u>L4</u>
<u>L3</u>	ingberg.xa. AND compiler	51	<u>L3</u>
<u>L2</u>	ingberg.xa. AND compiler and optimiz	0	<u>L2</u>
<u>L1</u>	ingberg.xa. AND knuth	0	<u>L1</u>

END OF SEARCH HISTORY

[Generate Collection](#)[Print](#)**Search Results - Record(s) 1 through 28 of 28 returned.** 1. Document ID: US 6629312 B1

L8: Entry 1 of 28

File: USPT

Sep 30, 2003

US-PAT-NO: 6629312

DOCUMENT-IDENTIFIER: US 6629312 B1

TITLE: Programmatic synthesis of a machine description for retargeting a compiler

DATE-ISSUED: September 30, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Gupta; Shail Aditya	Sunnyvale	CA		

US-CL-CURRENT: 717/136; 703/20, 717/147

ABSTRACT:

An MDES extractor automatically extracts a machine description (MDES) for re-targeting a compiler from a structural representation of a datapath of an explicitly parallel instruction computing (EPIC) processor. The datapath is a machine readable data structure that specifies the functional unit instances and an interconnect of the functional unit instances to registers. The MDES extractor structurally traverses the interconnect, identifying resource conflicts among the operations in the processor's opcode repertoire. Latencies and internal resources of the opcodes associated with the functional unit instances are obtained from a macrocell library. The MDES extractor then identifies external resource conflicts by preparing reservation tables for the functional units.

18 Claims, 17 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 9

[Full](#) | [Title](#) | [Citation](#) | [Front](#) | [Review](#) | [Classification](#) | [Date](#) | [Reference](#) | [Sequences](#) | [Attachments](#) | [Claims](#) | [RWD](#) | [Draw Desc](#) | [Image](#)
 2. Document ID: US 6578193 B1

L8: Entry 2 of 28

File: USPT

Jun 10, 2003

US-PAT-NO: 6578193

DOCUMENT-IDENTIFIER: US 6578193 B1

TITLE: Endian-neutral loader for interpretive environment

DATE-ISSUED: June 10, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Adams; Phillip M.	Salt Lake City	UT		

ABSTRACT:

A method is disclosed for a endian correction at load time, thereby eliminating the need to perform multiple endian correction routines during execution. The method comprises obtaining a platform endian context corresponding to the processor; obtaining a operand endian context indicating the ordering of operands contained in the set of instructions to be loaded; reading an instruction in the set of instructions; determining whether an operational code for the instruction is endian antithetical to the platform endian context; if the operational code for the instruction is endian antithetical, reversing the endian order of the instruction; loading the instruction into an appropriate memory location; and repeating the above steps as required for each instruction until all of the instructions have been loaded into memory.

16 Claims, 18 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 16

[Full](#) | [Title](#) | [Citation](#) | [Front](#) | [Review](#) | [Classification](#) | [Date](#) | [Reference](#) | [Sequences](#) | [Attachments](#) | [Claims](#) | [EPOC](#) | [Draw Desc](#) | [Image](#)

3. Document ID: US 6523168 B1

L8: Entry 3 of 28

File: USPT

Feb 18, 2003

US-PAT-NO: 6523168

DOCUMENT-IDENTIFIER: US 6523168 B1

TITLE: Reduction of object creation during string concatenation and like operations that utilize temporary data storage

DATE-ISSUED: February 18, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Arnold; Jeremy Alan	Rochester	MN		
Barsness; Eric Lawrence	Pine Island	MN		
Santosuoso; John Matthew	Rochester	MN		

ABSTRACT:

Reduction of object creation during string concatenation and like operations that utilize temporary data storage during translating a first computer program into a second computer program in which program code is generated to utilize a reusable temporary object in the performance of multiple operations that require the use of temporary storage, e.g., string concatenation operations. As such, the reusable temporary object need only be allocated once, in contrast with conventional implementations where multiple temporary objects would otherwise need to be allocated in the performance of such operations. Consequently, the additional overhead associated with allocating memory for additional objects, as well as collecting such objects once they are no longer in use, is eliminated, thereby improving overall performance while handling such multiple operations.

33 Claims, 5 Drawing figures

Exemplary Claim Number: 16

Number of Drawing Sheets: 3

[Full](#) | [Title](#) | [Citation](#) | [Front](#) | [Review](#) | [Classification](#) | [Date](#) | [Reference](#) | [Sequences](#) | [Attachments](#) | [EPOC](#) | [Draw Desc](#) | [Image](#)

4. Document ID: US 6499137 B1

L8: Entry 4 of 28

File: USPT

Dec 24, 2002

US-PAT-NO: 6499137

DOCUMENT-IDENTIFIER: US 6499137 B1

TITLE: Reversible load-time dynamic linking

DATE-ISSUED: December 24, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Hunt; Galen C.	Bellevue	WA		

US-CL-CURRENT: 717/164; 717/177

ABSTRACT:

A library links to a compiled application using the following variation of load-time dynamic linking. At some point prior to linking, a user selects a library for linking to the compiled application. An association is made between the selected library and any external libraries referenced within the compiled application. For example, if the application is in Common Object File format, a new import table lists the selected library and the external libraries of the original import table. At link time, the selected library and the external libraries link to the compiled application. At load time, the application, selected library, and any external libraries load. When the selected library loads first, a function in the selected library performs operations before the application or external libraries load. A pointer references the list of libraries to be linked to the compiled application. The initial state of this pointer is archived. The linking process becomes reversible by restoring the initial state of the pointer and re-linking. By replacing the reference to the selected library with a reference to a second selected table, a second selected library links to the application. A data record associated with the selected library enables additional functionality of the selected library.

48 Claims, 18 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 18

[Full](#) | [Title](#) | [Citation](#) | [Front](#) | [Review](#) | [Classification](#) | [Date](#) | [Reference](#) | [Sequences](#) | [Attachments](#)

[Image](#) | [Draw Desc](#) | [Image](#)

5. Document ID: US 6434740 B1

L8: Entry 5 of 28

File: USPT

Aug 13, 2002

US-PAT-NO: 6434740

DOCUMENT-IDENTIFIER: US 6434740 B1

TITLE: Apparatus and method for visual construction simplification

DATE-ISSUED: August 13, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Monday; Paul Brian	Rochester	MN		
Rubin; Bradley Scott	Rochester	MN		
Gavrilov; Galina	Riga			LV
Krasnikovs; Nikolajs	Riga			LV
Sulkins; Zahara	Riga			LV

US-CL-CURRENT: 717/108; 717/136

ABSTRACT:

A method and apparatus for connecting two components is disclosed. A visual construction simplification mechanism, as part of the apparatus, is designed with intelligence that allows it to correctly configure and interconnect the two components. The invention reduces the real and visual complexity of a program by requiring only a single bootstrap connection to be established between the components. Once the initial connection is made, the visual construction simplification mechanism can make additional connections that allow the two objects two function and interoperate. Introspection is a process used to find a component's interface and identity. The visual construction simplification mechanism looks at the interface of one component and figures out what the identity of the component is using the process of introspection. Introspection is a generic term for the ability of any component to look at the details of another component. Using the information gained through introspection, the visual construction simplification mechanism connects the correct methods and events to properly interconnect the two components.

52 Claims, 6 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 5

Full	Title	Citation	Front	Review	Classification	Date	Reference	Sequences	Attachments	Iconic	Draft Desc	Image
------	-------	----------	-------	--------	----------------	------	-----------	-----------	-------------	--------	------------	-------

6. Document ID: US 6405364 B1

L8: Entry 6 of 28

File: USPT

Jun 11, 2002

US-PAT-NO: 6405364

DOCUMENT-IDENTIFIER: US 6405364 B1

TITLE: Building techniques in a development architecture framework

DATE-ISSUED: June 11, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Bowman-Amuah; Michel K.	Colorado Springs	CO		

US-CL-CURRENT: 717/101; 717/102, 717/120, 717/124

ABSTRACT:

A system is provided for building systems in a development architecture framework. The present invention is directed to both a system to be built and an implementation strategy to fulfill system requirements. Software components of the system are encapsulated with wrappers. The wrappers are adapted to be changed upon other software components of the system being changed while the encapsulated software components of the system remain unchanged. In one embodiment of the present invention, specifying the requirements of the system to be built and the implementation strategy to fulfill the requirements may be carried out using tools such as data modeling tools, process modeling tools, event modeling tools, performance modeling tools, object modeling tools, component modeling tools, reuse support tools, prototyping tools, application logic design tools, database design

tools, presentation design tools, communication design, and usability test tools. In another embodiment of the present invention, improving the performance and maintenance of the system may be carried out using tools such as interactive navigation tools, graphical representation tools, extraction tools, repository tools, restructuring tools, and data name rationalization tools.

12 Claims, 14 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 14

[Full](#) | [Title](#) | [Citation](#) | [Front](#) | [Review](#) | [Classification](#) | [Date](#) | [Reference](#) | [Sequences](#) | [Attachments](#) | [KWC](#) | [Drawn Desc](#) | [Image](#)

7. Document ID: US 6397379 B1

L8: Entry 7 of 28

File: USPT

May 28, 2002

US-PAT-NO: 6397379

DOCUMENT-IDENTIFIER: US 6397379 B1

TITLE: Recording in a program execution profile references to a memory-mapped active device

DATE-ISSUED: May 28, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Yates, Jr.; John S.	Needham	MA		
Reese; David L.	Westborough	MA		
Van Dyke; Korbin S.	Sunol	CA		

US-CL-CURRENT: 717/140

ABSTRACT:

A method and a computer for execution of the method. As part of executing a stream of instructions, a series of memory loads is issued from a computer CPU to a bus, some directed to well-behaved memory and some directed to non-well-behaved devices in I/O space. Computer addresses are stored of instructions of the stream that issued memory loads to the non-well-behaved memory, the storage form of the recording allowing determination of whether the memory load was to well-behaved memory or not-well-behaved memory without resolution of any memory address stored in the recording.

47 Claims, 5 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 41

[Full](#) | [Title](#) | [Citation](#) | [Front](#) | [Review](#) | [Classification](#) | [Date](#) | [Reference](#) | [Sequences](#) | [Attachments](#) | [KWC](#) | [Drawn Desc](#) | [Image](#)

8. Document ID: US 6385769 B1

L8: Entry 8 of 28

File: USPT

May 7, 2002

US-PAT-NO: 6385769

DOCUMENT-IDENTIFIER: US 6385769 B1

TITLE: Text based object oriented program code with a visual program builder and parser support for predetermined and not predetermined formats

DATE-ISSUED: May 7, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Lewallen; Stephen	Cupertino	CA		

US-CL-CURRENT: 717/125, 717/116, 717/143

ABSTRACT:

Text-based object-oriented class code, located in either local or remote machines is converted into proxy components which can be used in existing visual builders. Proxy components are created from each method, including constructors, in the class code and encapsulate the parameters of the methods. For example, parameters associated with a method are represented by properties of the proxy component created from that method. These properties are visually editable and can be bound visually to other component properties using, for example, pull down menus in a visual builder. Exceptions which occur during operation of the method are treated as events and can be visually passed to other components. An add-on allows the components to appear directly in the visual builder palette. The components can interact with the text-based class code by means of a universal transport API.

30 Claims, 10 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 10

[Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments] [Drawn | Drawn Descr | Image]

9. Document ID: US 6311324 B1

L8: Entry 9 of 28

File: USPT

Oct 30, 2001

US-PAT-NO: 6311324

DOCUMENT-IDENTIFIER: US 6311324 B1

TITLE: Software profiler which has the ability to display performance data on a computer screen

DATE-ISSUED: October 30, 2001

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Smith; Kevin J.	Cupertino	CA		
Sridharan; K.	San Jose	CA		

US-CL-CURRENT: 717/114, 717/127, 717/130

ABSTRACT:

A C-language program performance tuning advisor that helps a systems analyst to improve the performance of an application. The tuning advisor identifies critical regions (hot spots) of an application, and helps the user to analyze the region. Once the region has been identified and analyzed, the tuning advisor advises the user on how to rewrite the original C code to improve the performance of the overall application. When the compiler needs to be conservative to be semantically correct, the tuning advisor suggests code modifications to remove the semantic constraints. The tuning advisor recognizes most commonly used C code patterns which if modified could improve the performance.

28 Claims, 4 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 3

10. Document ID: US 6272672 B1

L8: Entry 10 of 28

File: USPT

Aug 7, 2001

US-PAT-NO: 6272672

DOCUMENT-IDENTIFIER: US 6272672 B1

TITLE: Dataflow processing with events

DATE-ISSUED: August 7, 2001

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Conway; Melvin E.	Beverly	MA	01915	

US-CL-CURRENT: 717/107; 717/108

ABSTRACT:

Interactive event-driven programs are structured and executed using two types of constructs: interconnectable processing components and flow objects with associated data. Components are interconnected in a hierarchical dataflow network, and references which provide access to flow objects flow on the interconnections. Response to events and bidirectional coordination over multicomponent data paths, even in a distributed object system, employ unidirectional dataflows and intercomponent message sequences mediated by flow objects. Scaling and abstraction of complexity are facilitated by encapsulation of constructed networks into new component definitions. An interactive debugger preserves state as an executing program is edited, permitting an event-driven program to be modified in the intervals between processing of events without reinitialization. A component protection method employs multiple levels of usage authorization within components, enabling developers to define and distribute new protected components in a decentralized component market.

151 Claims, 115 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 77

 11. Document ID: US 6263492 B1

L8: Entry 11 of 28

File: USPT

Jul 17, 2001

US-PAT-NO: 6263492

DOCUMENT-IDENTIFIER: US 6263492 B1

TITLE: Run time object layout model with object type that differs from the derived object type in the class structure at design time and the ability to store the optimized run time object layout model

DATE-ISSUED: July 17, 2001

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Fraley; Christopher Lee	Woodinville	WA		
Halcoussis; Michael	Woodinville	WA		
Zimmerman; Christopher Alan	Bellevue	WA		
Carter; Alan W.	Bellevue	WA		
Wiltamuth; Scott Michael	Seattle	WA		
Burd; Gary S.	Kirkland	WA		
Hodges; C. Douglas	Redmond	WA		

US-CL-CURRENT: 717/107; 717/108

ABSTRACT:

A componentizing object designer is used to define a componentization of visual forms and other object-oriented technologies. The componentized object designer includes a set of tightly integrated protocols enabling Component Object Model (COM) objects to replace standard built-in visual form and other objects. The componentized object designer allows the design-time object and the run-time object to differ in implementation. The componentized object designer allows class identifiers for the run-time objects which are different than design-time objects. With a different class identifier, the run-time object can be saved as an object which is radically different from the design-time object. This enables the run-time object to be stored in a different object library than the design-time object. The componentized object designer allows for different persistence formats to be saved for run-time objects. The persistence formats for the run-time objects can be significantly smaller in size compared to the original design-time objects. This is important when the run-time object needs to be downloaded over a computer network like the Internet or an intranet.

38 Claims, 16 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 14

[Full](#) | [Title](#) | [Citation](#) | [Front](#) | [Review](#) | [Classification](#) | [Date](#) | [Reference](#) | [Sequences](#) | [Attachments](#)

[KOMC](#) | [Draw Desc](#) | [Image](#)

12. Document ID: US 6230314 B1

L8: Entry 12 of 28

File: USPT

May 8, 2001

US-PAT-NO: 6230314

DOCUMENT-IDENTIFIER: US 6230314 B1

TITLE: Method and device for program transformation using class hierarchy transformation based upon type constraint analysis

DATE-ISSUED: May 8, 2001

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Sweeney; Peter F.	Chestnut Ridge	NY		
Tip; Frank	Mount Kisco	NY		

US-CL-CURRENT: 717/108; 717/154

ABSTRACT:

A mechanism is provided that eliminates redundant components from objects of a program. Specifically, the mechanism is capable of detecting situations where a member of a given class is used by some, but not all instances of that class, and the elimination of this member from the instances where it is not needed. This is accomplished by an analysis of the program and its class hierarchy, followed by the

construction of a new, specialized class hierarchy and a transformation of the program. These operations preserve the original behavior of the program, and have the effect of "optimizing away" unneeded class members from objects. The invention is also capable of replacing class hierarchies that exhibit virtual inheritance with class hierarchies that only exhibit nonvirtual inheritance, and is applicable across a broad spectrum of inheritance structures. Transformation of virtual into nonvirtual inheritance improves program performance because it reduces the time required to access members that are located in virtual base classes. In addition, it may reduce the space required to represent objects.

20 Claims, 28 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 28

[Full](#) | [Title](#) | [Citation](#) | [Front](#) | [Review](#) | [Classification](#) | [Date](#) | [Reference](#) | [Sequences](#) | [Attachments](#) | [HTML](#) | [Drawn Desc](#) | [Image](#)

13. Document ID: US 6182283 B1

L8: Entry 13 of 28

File: USPT

Jan 30, 2001

US-PAT-NO: 6182283

DOCUMENT-IDENTIFIER: US 6182283 B1

TITLE: Linker optimization for compiled object oriented programs

DATE-ISSUED: January 30, 2001

INVENTOR-INFORMATION:

NAME

CITY

STATE

ZIP CODE

COUNTRY

Thomson; Brian Ward

North York

CA

US-CL-CURRENT: 717/153; 717/162

ABSTRACT:

When compiling a program using an object oriented language and virtual functions addressed by virtual function tables, the program may include virtual functions that are defined but not used. The compiler identifies such functions by tagging each defined virtual function with a code identifying it as a member of related virtual functions. The compiler also tags each virtual function call to identify which group (or groups) of related virtual functions are candidates for invocation by the virtual function call at runtime. The linker combines the two codes to identify which defined virtual functions are not candidates for invocation by any virtual function call. The linker omits those non-candidate virtual functions from the link if all references to them were from virtual function tables.

17 Claims, 12 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 12

[Full](#) | [Title](#) | [Citation](#) | [Front](#) | [Review](#) | [Classification](#) | [Date](#) | [Reference](#) | [Sequences](#) | [Attachments](#) | [HTML](#) | [Drawn Desc](#) | [Image](#)

14. Document ID: US 6161219 A

L8: Entry 14 of 28

File: USPT

Dec 12, 2000

US-PAT-NO: 6161219

DOCUMENT-IDENTIFIER: US 6161219 A

TITLE: System and method for providing checkpointing with precompile directives and

supporting software to produce checkpoints, independent of environment constraints

DATE-ISSUED: December 12, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Ramkumar; Balkrishna	Coralville	IA		
Strumpen; Volker	Boston	MA		

US-CL-CURRENT: 717/130; 717/116, 717/118, 717/158

ABSTRACT:

A method for portable checkpointing comprising the steps of: pre-compiling an input source code; and outputting an associated output source code which includes support for portable checkpointing. The portability of the checkpoints allows migration between systems with different hardware, software, and operating systems.

The portable checkpoints are stored in a Universal Code Format (UCF) and are selectively activated at the next potential checkpoint marker after a MinTBC (Minimum Time Between Checkpoints) timer expires.

51 Claims, 28 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 28

[Full](#) | [Title](#) | [Citation](#) | [Front](#) | [Review](#) | [Classification](#) | [Date](#) | [Reference](#) | [Sequences](#) | [Attachments](#)

[Draw](#) | [Desc](#) | [Image](#)

15. Document ID: US 6154877 A

L8: Entry 15 of 28

File: USPT

Nov 28, 2000

US-PAT-NO: 6154877

DOCUMENT-IDENTIFIER: US 6154877 A

TITLE: Method and apparatus for portable checkpointing using data structure metrics and conversion functions

DATE-ISSUED: November 28, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Ramkumar; Balkrishna	Coralville	IA		
Strumpen; Volker	Boston	MA		

US-CL-CURRENT: 717/114; 717/127

ABSTRACT:

A method and apparatus for portable checkpointing comprising the steps of: pre-compiling an input application source code and basic data type conversion functions; and outputting an associated output application source code and structure metrics and conversion function source code, wherein the output application source code includes support for portable checkpointing. The portability of the checkpoints allows migration between systems with different hardware, software, and operating systems. The present invention additionally provides a method and apparatus for conversion of data representations between a local machine format and a Universal Checkpoint Format (UCF) and a method for pointer handling, which involves transforming an absolute pointer into a machine independent offset and vice versa.

32 Claims, 46 Drawing figures

Exemplary Claim Number: 1

16. Document ID: US 6154876 A

L8: Entry 16 of 28

File: USPT

Nov 28, 2000

US-PAT-NO: 6154876

DOCUMENT-IDENTIFIER: US 6154876 A

TITLE: Analysis of the effect of program execution of calling components with data variable checkpointing and resource allocation analysis

DATE-ISSUED: November 28, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Haley; Matthew A.	San Jose	CA		
Pincus; Jonathan D.	San Francisco	CA		
Bush; William R.	San Mateo	CA		

US-CL-CURRENT: 717/133; 714/38, 717/126

ABSTRACT:

An error detection mechanism for detecting programming errors in a computer program. A component of the computer program, e.g., a procedure or function of the computer program, is analyzed to determine the effect of the component on resources used by the computer program. A component is analyzed by traversing the computer instructions, i.e., statements, of the component and tracking the state of resources used by the component as affected by the statements of the component. Each resource has a prescribed behavior represented by a number of states and transition between states. Violations in the prescribed behavior of a resource resulting from an emulated execution of the statements of the component are detected and reported as programming errors. Resources used by two or more components are modelled by modelling externals of the components. The effect of execution of a component on externals and resources of the component is determined by traversing one or more possible control flow paths through the component and tracking the use of each external and resource by each statement of each control flow path. Once the effect of execution of a component on externals and resources of the component is determined, a model of the component is created and used to model externals and resources of other components which invoke the modelled component.

5 Claims, 50 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 39

 17. Document ID: US 6151703 A

L8: Entry 17 of 28

File: USPT

Nov 21, 2000

US-PAT-NO: 6151703

DOCUMENT-IDENTIFIER: US 6151703 A

TITLE: Development system with methods for just-in-time compilation of programs

DATE-ISSUED: November 21, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Crelier; Regis	Santa Cruz	CA		

US-CL-CURRENT: 717/136

ABSTRACT:

A development system having a client which employs a virtual machine for executing programs written in the Java programming language is described. The client executes a "compiled" (i.e., bytecode or pseudo-compiled) Java program, which has been created by compiling a Java source code program or script with a Java compiler. The pseudo-compiled program comprises the bytecode emitted by the compiler. The development system further includes a just-in-time compiler which natively compiles each pseudo-compiled method of a Java program on a "just-in-time" basis--that is, compiles each method as it is actually used into native machine code for a target microprocessor. Methods which are unused are left uncompiled (i.e., left as bytecode). During program execution, when a method call is made from interpreted code, the system employs an "invoker" slot of the callee. When a method call is made from compiled code, the system employs a "compiled code" slot of the callee. As the addresses for the slots themselves remain unchanged, a method which has been compiled need not be recompiled when a callee method it invokes is itself compiled. In this manner, a method (caller) calling another method (callee) need not know whether it is calling is an interpreted method or a compiled method.

23 Claims, 8 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 7

[Full](#) | [Title](#) | [Citation](#) | [Front](#) | [Review](#) | [Classification](#) | [Date](#) | [Reference](#) | [Sequences](#) | [Attachments](#)

[PDF](#) | [Draw Desc](#) | [Image](#)

18. Document ID: US 6141792 A

L8: Entry 18 of 28

File: USPT

Oct 31, 2000

US-PAT-NO: 6141792

DOCUMENT-IDENTIFIER: US 6141792 A

TITLE: Object oriented framework for specifying the format of compiler output with a template facility

DATE-ISSUED: October 31, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Acker; Liane Elizabeth	Round Rock	TX		
Conner; Michael Haden	Austin	TX		
Martin; Andrew Richard	Austin	TX		

US-CL-CURRENT: 717/116; 717/140

ABSTRACT:

The format of an output file from a compiler is altered using two new objects, a template definition file for specifying the desired formats for the sections of the output file and a template facility for formatting output from the compiler according to the template definition file. The desired formats are specified by patterned sets of symbol names in the template definition file. A symbol table which is maintained by the template facility stores a set of symbol values corresponding to the symbol names. The template facility outputs the section by placing symbol

values from the symbol table in the output file according to the positions of the corresponding symbol names in the patterned sets of symbol names.

12 Claims, 14 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 12

[Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments] [KWMC | Drawn Desc | Image]

19. Document ID: US 6083282 A

L8: Entry 19 of 28

File: USPT

Jul 4, 2000

US-PAT-NO: 6083282
DOCUMENT-IDENTIFIER: US 6083282 A
** See image for Certificate of Correction **

TITLE: Cross-project namespace compiler and method

DATE-ISSUED: July 4, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Caron; Ilan Gabriel	Redmond	WA		
Carter; Alan W.	Bellevue	WA		
Canady; Dennis Mark	Redmond	WA		
Corbett; Tom	Eugene	OR		

US-CL-CURRENT: 717/101; 717/142

ABSTRACT:

In compiling source code organized in projects, unqualified name references are bound to program objects by searching namespaces in ascending hierarchical order from a current program unit to a current project. Namespaces within projects directly referenced by the current project are then searched. The search is completed at the first namespace in the search order which contains the matching name. Thus, program objects in projects directly referenced by the current project can be referenced by an unqualified name, without import or export lists of program elements.

24 Claims, 7 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 7

[Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments] [KWMC | Drawn Desc | Image]

20. Document ID: US 6072950 A

L8: Entry 20 of 28

File: USPT

Jun 6, 2000

US-PAT-NO: 6072950
DOCUMENT-IDENTIFIER: US 6072950 A

TITLE: Pointer analysis by type inference combined with a non-pointer analysis

DATE-ISSUED: June 6, 2000

INVENTOR-INFORMATION:

NAME Steensgaard; Bjarne	CITY North Bend	STATE WA	ZIP CODE	COUNTRY
-----------------------------	--------------------	-------------	----------	---------

US-CL-CURRENT: 717/126, 717/141, 717/151

ABSTRACT:

A pointer analysis by type inference combined with a non-pointer analysis helps approximate run-time store usage for a computer program. The analysis initially describes the content of each location for the program with a separate type as a non-pointer value. The analysis identifies store relationships described by the program and determines whether the location(s) and/or function(s) affected by the identified store relationships are well-typed under typing constraints. For well-typed store relationships, the analysis identifies any potential points-to relationships for types representing non-pointer values in case the analysis subsequently determines in processing other store relationships that the types may represent a pointer value. If the identified store relationships are not well-typed, the analysis modifies types for location(s) and/or function(s) affected by the identified store relationships as necessary so the store relationships are well-typed. The

analysis also modifies types for locations and/or functions for potential points-to relationships affected by the modification of types. When the locations and/or functions for all identified store relationships are well-typed, the program is well-typed with the set of types defining a store model for the program.

56 Claims, 6 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 6

[Full](#) | [Title](#) | [Citation](#) | [Front](#) | [Review](#) | [Classification](#) | [Date](#) | [Reference](#) | [Sequences](#) | [Attachments](#)

[EPOC](#) | [Draw Desc](#) | [Image](#)

21. Document ID: US 6047125 A

L8: Entry 21 of 28

File: USPT

Apr 4, 2000

US-PAT-NO: 6047125

DOCUMENT-IDENTIFIER: US 6047125 A

TITLE: Garbage collection system for improved use of memory by removal of reference conflicts

DATE-ISSUED: April 4, 2000

INVENTOR-INFORMATION:

NAME Agesen; Ole Detlefs; David L.	CITY Franklin Westford	STATE MA MA	ZIP CODE	COUNTRY
--	------------------------------	-------------------	----------	---------

US-CL-CURRENT: 717/148, 707/201, 707/206, 709/100, 709/106, 709/315, 711/100,
711/103, 711/165, 711/206, 712/216, 717/154

ABSTRACT:

In accordance with the present invention a method for modifying a sequence of instructions to improve memory management within a storage device during execution of the instructions, comprises the steps, performed by a processor, of (a) analyzing the sequence of instructions for a conflict indicating an undeterminable variable type, (b) determining the type of conflict, and (c) modifying the sequence of instructions to eliminate the conflict based on the determination.

29 Claims, 11 Drawing figures
Exemplary Claim Number: 1

Number of Drawing Sheets: 10

[Full](#) | [Title](#) | [Citation](#) | [Front](#) | [Review](#) | [Classification](#) | [Date](#) | [Reference](#) | [Sequences](#) | [Attachments](#) | [KMC](#) | [Draw Desc](#) | [Image](#)

22. Document ID: US 6044216 A

L8: Entry 22 of 28

File: USPT

Mar 28, 2000

US-PAT-NO: 6044216

DOCUMENT-IDENTIFIER: US 6044216 A

TITLE: Method and apparatus for implementing cursor variables for accessing data from database

DATE-ISSUED: March 28, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Bhargava; Sunil	San Carlos	CA		
Peschansky; Olga	Cupertino	CA		

US-CL-CURRENT: 717/114; 717/126

ABSTRACT:

A method and apparatus for implementing a data construct, referred to herein as a "cursor variable", that has many of the attributes of simple variables and that can be used to access active sets of data from a database. Cursor variables identify a current set of data in an active set generated as a result of a database query. A cursor variable can be associated with any one of a number of different queries at any given time during program execution without using a host programming language. As a result, different active sets having different columns, different tables, and/or different predicates may be associated with the same cursor variable using instructions from a single programming language.

51 Claims, 13 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 11

[Full](#) | [Title](#) | [Citation](#) | [Front](#) | [Review](#) | [Classification](#) | [Date](#) | [Reference](#) | [Sequences](#) | [Attachments](#) | [KMC](#) | [Draw Desc](#) | [Image](#)

23. Document ID: US 6029002 A

L8: Entry 23 of 28

File: USPT

Feb 22, 2000

US-PAT-NO: 6029002

DOCUMENT-IDENTIFIER: US 6029002 A

TITLE: Method and apparatus for analyzing computer code using weakest precondition

DATE-ISSUED: February 22, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Afifi; Ashraf	Burlington	MA		
Chan; Dominic	Carlisle	MA		
Comuzzi; Joseph J.	Groton	MA		
Hart; Johnson M.	Weston	MA		
Pizzarello; Antonio	Phoenix	AZ		

US-CL-CURRENT: 717/131; 717/125, 717/144, 717/146

ABSTRACT:

An analyzer for maintaining and analyzing source code is disclosed. The analyzer includes a software translator for converting conventional source code into an intermediate language, slicing capability based upon weakest precondition determination, dual direction flow analysis and incorporation of a computational model to facilitate iterative code.

22 Claims, 95 Drawing figures
 Exemplary Claim Number: 1
 Number of Drawing Sheets: 63

[Full](#) | [Title](#) | [Citation](#) | [Front](#) | [Review](#) | [Classification](#) | [Date](#) | [Reference](#) | [Sequences](#) | [Attachments](#) | [HTMLC](#) | [Drawn Desc](#) | [Image](#)

24. Document ID: US 6023582 A

L8: Entry 24 of 28

File: USPT

Feb 8, 2000

US-PAT-NO: 6023582
 DOCUMENT-IDENTIFIER: US 6023582 A

TITLE: Function code chaining method

DATE-ISSUED: February 8, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Rogers; Norman L.	Davis	CA		
Auyeung; Tak Y.	Sacramento	CA		

US-CL-CURRENT: 717/162; 717/143

ABSTRACT:

A computer language construct for connecting related but independent routines at compile time. The computer language construct allows the creation of symbolic "chains", which can be called at run time. Individual functions and code fragments, written in a high level programming language, can be attached to the chains. When a symbolic chain is called, all the functions and code fragments attached to it will also be called. The definition of a symbolic chain and each function or code fragment attached to it are all independent. Chains can also be organized hierarchically, and a chain can be attached to another chain.

13 Claims, 7 Drawing figures
 Exemplary Claim Number: 1
 Number of Drawing Sheets: 4

[Full](#) | [Title](#) | [Citation](#) | [Front](#) | [Review](#) | [Classification](#) | [Date](#) | [Reference](#) | [Sequences](#) | [Attachments](#) | [HTMLC](#) | [Drawn Desc](#) | [Image](#)

25. Document ID: US 6011919 A

L8: Entry 25 of 28

File: USPT

Jan 4, 2000

US-PAT-NO: 6011919

DOCUMENT-IDENTIFIER: US 6011919 A

** See image for Certificate of Correction **

TITLE: Method of providing efficiency to a graphical programming language with alternative form determination and cost of execution estimation

DATE-ISSUED: January 4, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Politis; George	Macquarie Fields			AU
Long; Timothy Merrick	Lindfield			AU

US-CL-CURRENT: 717/114; 717/139

ABSTRACT:

A system, method and language for compositing or creating images is disclosed. The images typically comprise a plurality of graphical elements each including color and opacity information. The system utilizes operators having the graphical elements as operands in which the operators combine the operands according to a function defined by the operators, the colour information, and the opacity information, to produce new graphical elements. One part of the system includes interpreting the language by parsing and executing a sequence of statements and forming an expression tree the nodes of which comprise the graphical elements. Instructions are then derived from the tree. Another part permits the compositing of opaque graphical elements and associated clipping operations. Bounding box methods are used for locating active areas of graphical elements from the nodes. Manipulation of the expression tree is used to reduce the expected execution time of the compositing commands. An architecture is disclosed for implementing the system.

31 Claims, 35 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 15

[Full](#) | [Title](#) | [Citation](#) | [Front](#) | [Review](#) | [Classification](#) | [Date](#) | [Reference](#) | [Sequences](#) | [Attachments](#)

[RDMC](#) | [Draw Desc](#) | [Image](#)

26. Document ID: US 6002875 A

L8: Entry 26 of 28

File: USPT

Dec 14, 1999

US-PAT-NO: 6002875

DOCUMENT-IDENTIFIER: US 6002875 A

TITLE: Method for the reduction of instruction cache miss rate using optimization data from trace data profiles

DATE-ISSUED: December 14, 1999

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Stolberg; Hans-Joachim	Tokyo			JP

US-CL-CURRENT: 717/153

ABSTRACT:

A method for the reduction of instruction cache misses comprises step of generating function profiles of potential cache misses, step of computing function activities and dividing them in nonzero-activity functions and zero-activity functions, step of allocating all the nonzero-activity functions to cache space, step of mapping the nonzero-activity functions from cache space to memory space, step of mapping the zero-activity functions so as to fill allocation holes in memory space.

12 Claims, 9 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 9

[Full](#) | [Title](#) | [Citation](#) | [Front](#) | [Review](#) | [Classification](#) | [Date](#) | [Reference](#) | [Sequences](#) | [Attachments](#)

[HTMLC](#) | [Draw Desc](#) | [Image](#)

27. Document ID: US 5999737 A

L8: Entry 27 of 28

File: USPT

Dec 7, 1999

US-PAT-NO: 5999737

DOCUMENT-IDENTIFIER: US 5999737 A

TITLE: Link time optimization via dead code elimination, code motion, code partitioning, code grouping, loop analysis with code motion, loop invariant analysis and active variable to register analysis

DATE-ISSUED: December 7, 1999

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Srivastava; Amitabh	Menlo Park	CA		

US-CL-CURRENT: 717/162; 717/146, 717/159

ABSTRACT:

A computer system is directed to convert a program written as a plurality of high level source code modules into corresponding machine executable code. The source code modules are compiled into an object code module, and the object code modules are translated into a single linked code module in the form of a register translation language and logical symbol table compatible with a plurality of computer system hardware architectures. The source code program structures are recovered from the linked code module, and the linked code module is partitioned into a plurality of procedure, and instructions of each of the procedures grouped into basic blocks. A procedure flow graph is constructed for each of the procedures, and a program call graph is constructed for the linked code module. The linked code module is modified by eliminating dead code and moving loop-invariant code from loops. The modified linked code is converted into machine executable code compatible with a target one of said plurality of computer system hardware architectures.

23 Claims, 9 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 9

[Full](#) | [Title](#) | [Citation](#) | [Front](#) | [Review](#) | [Classification](#) | [Date](#) | [Reference](#) | [Sequences](#) | [Attachments](#)

[HTMLC](#) | [Draw Desc](#) | [Image](#)

28. Document ID: US 5966539 A

L8: Entry 28 of 28

File: USPT

Oct 12, 1999

US-PAT-NO: 5966539

DOCUMENT-IDENTIFIER: US 5966539 A

TITLE: Link time optimization with translation to intermediate program and following optimization techniques including program analysis code motion live variable set generation order analysis, dead code elimination and load invariant analysis

DATE-ISSUED: October 12, 1999

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Srivastava; Amitabh	Menlo Park	CA		

US-CL-CURRENT: 717/156; 717/147, 717/160

ABSTRACT:

A computer system is directed to convert a program written as a plurality of high level source code modules into corresponding machine executable code. The source code modules are compiled into an object code module, and the object code modules are translated into a single linked code module in the form of a register translation language and logical symbol table compatible with a plurality of computer system hardware architectures. The source code program structures are recovered from the linked code module, and the linked code module is partitioned into a plurality of procedure, and instructions of each of the procedures grouped into basic blocks. A procedure flow graph is constructed for each of the procedures, and a program call graph is constructed for the linked code module. The linked code module is modified by eliminating dead code and moving loop-invariant code from loops. The modified linked code is converted into machine executable code compatible with a target one of said plurality of computer system hardware architectures.

21 Claims, 9 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 9

[Full](#) | [Title](#) | [Citation](#) | [Front](#) | [Review](#) | [Classification](#) | [Date](#) | [Reference](#) | [Sequences](#) | [Attachments](#) | [ViewC](#) | [Draw Desc](#) | [Image](#)

[Generate Collection](#) | [Print](#)

Terms	Documents
L7 and table	28

[Display Format:](#) [REV](#) | [Change Format](#)

[Previous Page](#) | [Next Page](#)